Übungen zum Kapitel 7

Datenaufbereitung - Säubern und Transformieren

Erstellt und überarbeitet: armin.baenziger@zhaw.ch, 21. Februar 2020

In [1]:
```
%autosave 0
```

```
Autosave disabled
```

(A.1) Laden Sie NumPy und Pandas mit der üblichen Abkürzung.

In [2]:
```
import numpy as np
import pandas as pd
```

(A.2) Laden Sie die Daten der Pickle-Datei `dflohn.pkl` mit `pd.read_pickle` in das DataFrame `dflohn` und lesen Sie die ersten 5 Zeilen aus. Die Datei befindet sich im Ordner "weitere_Daten".

In [3]:
```
dflohn = pd.read_pickle('../weitere_Daten/dflohn.pkl')
dflohn.head()
```

Out[3]:

Person	Lohn	Geschlecht	Alter	Zivilstand
1	4107.0	m	40	g
2	5454.0	m	47	vw
3	3719.0	m	41	g
4	6194.0	m	18	v
5	NaN	m	27	v

(A.3) Ermitteln Sie, wie viele `NaN` es gibt pro Spalte/Variable. Tipp: Die Methode `isnull()` gibt `True` zurück, falls `NaN` und ansonsten `False`. Die Methode `sum()` summiert `True` (1) und `False` (0) auf.

In [4]:
```
dflohn.isnull().sum()
# Es gibt nur einen Fehlwert in der Variable Lohn.
```

Out[4]:
```
Lohn          1
Geschlecht    0
Alter         0
Zivilstand    0
dtype: int64
```

(A.4) Ermitteln Sie, *welchen Anteil* an `NaN` es gibt pro Spalte/Variable. Tipp: Der Mittelwert aus den `False` - und `True` -Werten ergibt den gewünschten Anteilswert.

In [5]:
```
dflohn.isnull().mean()
```

Out[5]:
```
Lohn          0.01
Geschlecht    0.00
Alter         0.00
Zivilstand    0.00
dtype: float64
```

(A.5) Lesen Sie die ersten 5 Zeilen aus `dflohn` aus, welche *keinen* Fehlwert haben. Tipp: Methode `dropna` und danach Methode `head`.

In [6]:
```
dflohn.dropna().head()
```

Out[6]:

Person	Lohn	Geschlecht	Alter	Zivilstand
1	4107.0	m	40	g
2	5454.0	m	47	vw
3	3719.0	m	41	g
4	6194.0	m	18	v
6	4623.0	w	38	vw

(A.6) Lesen Sie die ersten 5 Zeilen aus `dflohn` aus, wobei Fehlwerte mit dem Wert 0 ersetzt werden sollen. Tipp: Methode `fillna`.

In [7]:
```
dflohn.fillna(0).head()
```

Out[7]:

Person	Lohn	Geschlecht	Alter	Zivilstand
1	4107.0	m	40	g
2	5454.0	m	47	vw
3	3719.0	m	41	g
4	6194.0	m	18	v
5	0.0	m	27	v

(A.7) Lesen Sie die ersten 5 Zeilen aus `dflohn` aus, wobei Fehlwerte mit dem Median (der verfügbaren Werte) in der entsprechenden Spalte ersetzt werden sollen. Tipp: Methode `fillna`.

In [8]:
```
dflohn.fillna(dflohn.median()).head()
```

Out[8]:

Person	Lohn	Geschlecht	Alter	Zivilstand
1	4107.0	m	40	g
2	5454.0	m	47	vw
3	3719.0	m	41	g
4	6194.0	m	18	v
5	5314.0	m	27	v

(A.8) Prüfen Sie, ob es Fälle (Zeilen) gibt, die mehr als einmal vorkommen. Tipp: Methode `duplicated`.

In [9]:
```
dflohn.duplicated().sum()
# Es gibt keine mehrfachen Zeilen.
```

Out[9]:
```
0
```

In [10]:
```
# Oder:
dflohn.index.is_unique
# Da der Index (Personen-ID) "einzigartig" ist, sind alle Zeilen
# unterschiedlich.
```

Out[10]:
```
True
```

(B.1) Ermitteln Sie alle unterschiedlichen Ausprägungen des Merkmals (Spalte) `Zivilstand` in `dflohn`. Tipp: Methode `unique()` verwenden oder die Funktion `set()`.

In [11]:
```
# Lösung 1:
dflohn.Zivilstand.unique()
```

Out[11]:
```
array(['g', 'vw', 'v', 'l'], dtype=object)
```

In [12]:
```
# Lösung 2:
set(dflohn.Zivilstand)
```

Out[12]:
```
{'g', 'l', 'v', 'vw'}
```

(B.2) Erstellen Sie eine Kopie des DataFrame `dflohn` mit Name `dflohn2`.

In [13]:
```
dflohn2 = dflohn.copy()
dflohn2.head(3)
```

Out[13]:

Person	Lohn	Geschlecht	Alter	Zivilstand
1	4107.0	m	40	g
2	5454.0	m	47	vw
3	3719.0	m	41	g

(B.3) Ändern Sie die Ausprägungen von `Zivilstand` *in der Kopie* `dflohn2` wie folgt:

- `v` in `verh`
- `vw` in `verw`
- `l` und `g` bleiben gleich

Verwenden Sie hierzu die Methode `replace()` mit einem Dict, welches die Zuordnung enthält.

In [14]:
```
zuordnung = {'v': 'verh', 'vw': 'verw'}
# Hinweis: Nur die zu ändernden Ausprägungen sind nötig.
dflohn2.Zivilstand.replace(zuordnung, inplace=True)
dflohn2.head()
```

Out[14]:

Person	Lohn	Geschlecht	Alter	Zivilstand
1	4107.0	m	40	g
2	5454.0	m	47	verw
3	3719.0	m	41	g
4	6194.0	m	18	verh
5	NaN	m	27	verh

(B.4) Ändern Sie im DataFrame `dflohn2` (der Kopie!) den Spaltennamen `Lohn` in `Monatslohn`. Tipp: Methode `rename()` mit Argument `columns` verwenden.

In [15]:
```
dflohn2.rename(columns={'Lohn': 'Monatslohn'}, inplace=True)
dflohn2.head()
```

Out[15]:

Person	Monatslohn	Geschlecht	Alter	Zivilstand
1	4107.0	m	40	g
2	5454.0	m	47	verw
3	3719.0	m	41	g
4	6194.0	m	18	verh
5	NaN	m	27	verh

Die folgenden Anweisungen teilen das Alter in Altersklassen ein.

In [16]:
```
klassen = range(15, 66, 10)
dflohn['Altersklasse'] = pd.cut(dflohn.Alter, klassen)
dflohn.head()
```

Out[16]:

Person	Lohn	Geschlecht	Alter	Zivilstand	Altersklasse
1	4107.0	m	40	g	(35, 45]
2	5454.0	m	47	vw	(45, 55]
3	3719.0	m	41	g	(35, 45]
4	6194.0	m	18	v	(15, 25]
5	NaN	m	27	v	(25, 35]

(C.1) Erstellen Sie eine (absolute) Häufigkeitsverteilung der Variable `Altersklasse`.

```
In [17]: dflohn.Altersklasse.value_counts()
```

```
Out[17]: (15, 25]    24
         (45, 55]    21
         (25, 35]    21
         (55, 65]    18
         (35, 45]    16
         Name: Altersklasse, dtype: int64
```

(C.2) Sortieren Sie obige Häufigkeitstabelle nach den Altersklassen (bzw. *nicht* nach den Häufigkeiten). Tipp: Argument `sort=False` oder Methode `sort_index`.

```
In [18]: dflohn.Altersklasse.value_counts(sort=False)
```

```
Out[18]: (15, 25]    24
         (25, 35]    21
         (35, 45]    16
         (45, 55]    21
         (55, 65]    18
         Name: Altersklasse, dtype: int64
```

```
In [19]: # Alternative Lösung:
         dflohn.Altersklasse.value_counts().sort_index()
```

```
Out[19]: (15, 25]    24
         (25, 35]    21
         (35, 45]    16
         (45, 55]    21
         (55, 65]    18
         Name: Altersklasse, dtype: int64
```

(C.3) Erstellen Sie eine *relative* Häufigkeitsverteilung der Variable `Altersklasse`, wobei nach Altersklasse sortiert sein soll. Tipp: Zusätzlich das Argument `normalize=True` verwenden.

```
In [20]: dflohn.Altersklasse.value_counts(sort=False, normalize=True)
```

```
Out[20]: (15, 25]    0.24
         (25, 35]    0.21
         (35, 45]    0.16
         (45, 55]    0.21
         (55, 65]    0.18
         Name: Altersklasse, dtype: float64
```

(D.1) Ziehen Sie eine Zufallsstichprobe (ohne Zurücklegen) aus `dflohn`, welche 5% der ursprünglichen Beobachtungen (Zeilen) enthält. Hinweis: `dflohn.sample()` und dann Shift-Tab ausführen.

In [21]:
```
dflohn.sample(frac=0.05)
```

Out[21]:

Person	Lohn	Geschlecht	Alter	Zivilstand	Altersklasse
96	7959.0	w	55	v	(45, 55]
91	7251.0	w	49	v	(45, 55]
35	3967.0	w	32	l	(25, 35]
68	9928.0	w	57	l	(55, 65]
46	4004.0	w	25	g	(15, 25]

(D.2) Erstellen Sie die Dummy-Variable `Geschlecht_w` im DataFrame `dflohn`, welche 0 ist, wenn es sich um einen Mann handelt (`m`) und 1, wenn es sich um eine Frau handelt (`w`).

In [22]:
```
# Verschiedene Möglichkeiten, z.B.:
pd.get_dummies(dflohn, columns=['Geschlecht'], drop_first=True).head(6)
# Am einfachsten, aber hier verschwindet die ursprüngliche
# Variable Geschlecht, was oft unerwünscht ist.
```

Out[22]:

Person	Lohn	Alter	Zivilstand	Altersklasse	Geschlecht_w
1	4107.0	40	g	(35, 45]	0
2	5454.0	47	vw	(45, 55]	0
3	3719.0	41	g	(35, 45]	0
4	6194.0	18	v	(15, 25]	0
5	NaN	27	v	(25, 35]	0
6	4623.0	38	vw	(35, 45]	1

In [23]:
```
# Oder so:
dflohn['Geschlecht_w'] = pd.get_dummies(dflohn.Geschlecht)['w']
dflohn.head(6)
```

Out[23]:

Person	Lohn	Geschlecht	Alter	Zivilstand	Altersklasse	Geschlecht_w
1	4107.0	m	40	g	(35, 45]	0
2	5454.0	m	47	vw	(45, 55]	0
3	3719.0	m	41	g	(35, 45]	0
4	6194.0	m	18	v	(15, 25]	0
5	NaN	m	27	v	(25, 35]	0
6	4623.0	w	38	vw	(35, 45]	1

In [24]:
```
# Oder auch so:
dflohn['Geschlecht_w'] = (dflohn.Geschlecht == 'w').astype(int)
dflohn.head(6)
```

Out[24]:

Person	Lohn	Geschlecht	Alter	Zivilstand	Altersklasse	Geschlecht_w
1	4107.0	m	40	g	(35, 45]	0
2	5454.0	m	47	vw	(45, 55]	0
3	3719.0	m	41	g	(35, 45]	0
4	6194.0	m	18	v	(15, 25]	0
5	NaN	m	27	v	(25, 35]	0
6	4623.0	w	38	vw	(35, 45]	1

Bevor wir mit den Stringmanipulationen beginnen, erzeugen wir einen Beispiel-String:

In [25]:
```
zeichenkette = 'peter_vogt@test.com, m.mueller@test.com, banz(at)zhaw.ch'
```

(E.1) Wie viele `test.com` -Email-Adressen sind im String `zeichenkette` enthalten? Tipp: Methode `count()` verwenden.

In [26]:
```
zeichenkette.count('test.com')
```

Out[26]:
```
2
```

(E.2) Erzeugen Sie die *Liste* `emails` mit den Email-Adressen aus dem String `zeichenkette` . Verwenden Sie hierzu die Methode `split` .

In [27]:
```
emails = zeichenkette.split(', ')
emails
```

Out[27]:
```
['peter_vogt@test.com', 'm.mueller@test.com', 'banz(at)zhaw.ch']
```

(E.3) Prüfen Sie, ob m.mueller@test.com in der Liste `emails` vorhanden ist.

In [28]:
```
'm.mueller@test.com' in emails
```

Out[28]:
```
True
```

(E.4) Transformieren Sie die Liste `emails` in die Series `emails` .

In [29]:
```
emails = pd.Series(emails)
emails
```

Out[29]:
```
0    peter_vogt@test.com
1     m.mueller@test.com
2        banz(at)zhaw.ch
dtype: object
```

(E.5) Alle `(at)` in der Series `emails` sollen mit `@` ersetzt werden. Studieren Sie hierzu folgende Anweisung:

In [30]:
```
emails.str.replace('\(at\)', '@')
# \ ist der "Escape-Character".
# Er bewirkt, dass "(" und ")" als Strings aufgefasst werden.
```

Out[30]:
```
0    peter_vogt@test.com
1     m.mueller@test.com
2           banz@zhaw.ch
dtype: object
```

(E.6) Wie viele Email-Adressen in der Series `emails` enden mit ".ch"? Tipp: Methode `str.endswith()` verwenden.

In [31]:
```
emails.str.endswith('.ch').sum()
```

Out[31]:
```
1
```

Ende der Übung